LOSSLESS DATA COMPRESSION FOR ENERGY EFFICIENT TRANSMISSION OVER WIRELESS NETWORK

Prof. Saurabh Mittal
Area Chairperson, Asia-Pacific Institute of Management, New Delhi
Email: Saurabh.m@asiapacific.edu, sau275@gmail.com

Abstract

Wireless networks are very fast replacing the traditional wired networking for the small and long distance area. With the rapid expansion in the data generation in the digital world and need for information sharing between the different stakeholders / collaborative members for business and scientific needs, today the volume of data being transmitted is also facing exponential growth. Wireless transmission of a single bit can require over 1000 times more energy than a single 32-bit computation. It can therefore be beneficial to perform additional computation to reduce the number of bits transmitted. If the energy required to compress data is less than the energy required to send it, there is a net energy savings and an increase in battery life for portable computers. This paper proposes energy savings possible by losslessly compressing data prior to transmission. The proposed work focuses that, with several typical compression algorithms, there is actually a net energy increase when compression is applied before transmission. Reasons for this increase are explained and proposal will be to avoid it. Key words: Compression, Lossless, Transmission, Wireless, Data.

Wireless communication is an essential component of mobile computing, but the energy required for transmission of a single bit has been measured to be over 1000 times greater than for a single 32-bit computation. Thus, if 1000 computation operations can compress data by even 1 bit, energy should be saved. Compression algorithms which once seemed too resource or time-intensive might be valuable for saving energy. Implementations which made concessions in compression ratio to improve performance might be modified to provide an overall energy saving. Ideally, the effort exerted to compress data should be variable to allow a flexible tradeoff between speed and energy. Earlier work has considered lossy compression techniques which sacrifice the quality of compressed audio or video data streams to reduce the bit rate and energy required. In proposed work, we consider the challenge of reducing wireless communication energy for data that must be transmitted faithfully. We will provide detailed survey of the energy requirements of several lossless data compression schemes. Several families of compression algorithms are analyzed and characterized, and it is shown that compression prior to transmission may actually cause an overall energy increase. We will focus on behaviors and resource usage patterns which allow for energy-efficient lossless compression of data. When applied to Unix compress, these optimizations improve energy efficiency by 51%. We also explore the fact that, for many usage models, compression and decompression need not be performed by the same algorithm. By choosing the lowest-energy compressor and decompressor on the test platform, rather than using default levels of compression, overall energy to send compressible web data can be reduced 31%. Energy to send harder-to-compress English text can be reduced 57%. Compared with a system using a single optimized application for both compression and decompression, the asymmetric scheme saves 11% or 12% of the total energy depending on the dataset. Proposed work focuses on asymmetric compression i.e. the use of one compression algorithm on the transmit side and a different algorithm for the receive path.

Compression Steps

Compression is usually broken into two steps: modeling and coding. With a perfect, concise model that describes the generation of the input source which is to be compressed, one could reproduce the data without transmitting the source data. (i.e., if the sequence 1 1 2 3 5 • • • 6765 were to be transmitted, one could express it with a “model” of Fibonacci numbers). In practice, one must approximate and construct an approximate mathematical model for the data. In English text, for example, one can model the probability of a letter occurring as a probability conditioned on letters that have already been transmitted. This probabilistic model
is transmitted with a description of how the data differs from the model.

In a coding step, information is mapped to compact code-words. Obviously, a codeword must decode to a unique value so there can be no doubt of the original message. Prefix codes are used so that no codeword is the prefix of any other codeword. It has been proved that, for any non-prefix code that may be uniquely decoded, a prefix code can be found with the same code word lengths. Often the modeling and coding steps are tightly coupled. For instance, Lempel-Ziv codes can be constructed as an input source is parsed into a “dictionary” model, when it is difficult to extricate the coding from the modeling.

Coding Methods

Coding maps symbols from the input alphabet into compact binary sequences.

1. Huffman Coding. If the probability of each source symbol is known a priori (perhaps by scanning through the source), a procedure known as static Huffman coding can be used to build an optimal code in which the most frequently occurring symbols are given the shortest codewords. Huffman codes are established by storing the symbols of the alphabet in a binary tree according to their probability. As the tree is traversed from root to leaf, the code grows in length. When visiting the right child, a 0 is appended to the code. When visiting the left child, a 1 is appended. Thus, symbols which occur frequently are stored near the root of the tree and have the shortest codes. Since data compression tools rarely have the luxury of a priori knowledge and cannot afford two passes through the data source, variants of the Huffman algorithm have been developed that work dynamically and update the tree as source symbols are encountered.

2. Arithmetic Coding. Optimal compression ratio for a data source is traditionally described with respect to Claude Shannon’s definition of source entropy: a measure of the source’s information and therefore the average number of bits required to represent it. Sometimes the most frequently occurring symbol can contain so little information that it would be ideal to represent it with less than 1 bit. Huffman codes are restricted to using an integral number of bits per symbol, increasing the coding overhead. Arithmetic codes have been designed to support a fractional number of bits per symbol to bring the average length of a codeword much closer to the optimal. Knowing the probability of occurrence for each symbol, a unique identifier can be established for a series of symbols. This identifier is a binary fraction in the interval [0,1]. Unlikely symbols narrow this interval so that more bits are required to specify it, while highly likely symbols add little information to a message and require the addition of fewer bits as the interval refinement is coarser. As the fraction converges, the most significant bits become fixed, so the fraction can be transmitted most-significant-bit-first as soon as it is known. Arithmetic coding requires frequent division and multiplication.

3. Lempel-Ziv Codes. A Lempel-Ziv codebook is made up of fixed-length code-words in which each entry has nearly the same probability of appearing, but in which longer groups of symbols are represented in the same length as single symbols. Thus, it may require extra bits to send the coded version of a single symbol, but a string of frequently occurring symbols can be represented with a fraction of the bits ordinarily required. Since only \(n\) code-words can be represented with \(\log(n)\) bits, systems for dynamically increasing the length of code-words exist.

Lossless Compression Algorithms

The coding techniques described above are used in the algorithm families introduced below. There are two fundamental methods for constructing Lempel-Ziv codes. Introduced in the late 1970s, these methods are known by the initials of their creators and the year of introduction: LZ77 and LZ78. Prediction with Partial Match (PPM) uses Markov modeling followed by arithmetic coding. The Burrows-Wheeler Transform (BWT) reversibly permutes a block of source data so that it can easily be compressed.

- **Sliding Window—LZ77.** LZ77 maintains a current pointer into the source data, a search buffer, and a look-ahead buffer. The search buffer is made up of symbols encountered prior to the current symbol, and the look-ahead buffer contains symbols which appear after the current symbol. Together, the buffers comprise a “window” which specifies the section of the input source under consideration. As the current pointer advances, the window “slides” over the input. As symbols are encountered in the look-ahead buffer, the algorithm searches backward for the longest match in the search buffer. Instead of transmitting the matched symbols, they can be encoded with a triple: <offset from pointer, length of match, next codeword>. The “next codeword” is the codeword corresponding to the symbol in the look-ahead buffer following the match. It is necessary in case a match for the look-ahead buffer
cannot be found. This scheme can be enhanced by using a variable length coder (e.g., Huffman coding) to reduce the size of the fixed-length triples. Another popular enhancement involves a more efficient way to represent a single character without an entire triple, using a flag to indicate whether a literal or match is being transmitted.

- **Dictionary—LZ78.** The LZ78 scheme was introduced to account for cases in which a nearby match cannot be found. Instead of the sliding search-buffer, LZ78 uses a separate dictionary, which also serves as a codebook. As each group of symbols is encountered, the dictionary is checked. An `<index, code>` pair is output where *index* corresponds to the longest prefix (if any) that matches the current input, and *code* is the unmatched symbol which follows. The pair is then added to the dictionary. The decompressor builds its dictionary in a corresponding fashion so that received indices refer to the same symbol as they did in the compressor. A popular improvement to LZ78 is called LZW. It seeds the dictionary with letters from the source alphabet which eliminates the need to send the second element of the pair, shortening the number of bits that must be sent for a single character. With every symbol present in the dictionary, only the index need be sent. Since each new dictionary entry contains a pointer to a previous entry, decoding occurs recursively, requiring decompression to buffer symbols in a stack and reverse them before output. Such a system results in the quick accumulation of long patterns which can be stored indefinitely, but has several drawbacks. Until the dictionary is filled with longer frequently seen patterns, “compressed” output will be larger than in its original form. Since the dictionary can grow without bound, implementations of LZ78 must erase the dictionary when it gets too large, freeze the dictionary and continue in a nonadaptive fashion, or adopt another policy to limit memory usage.

- **Prediction with Partial Match—PPM.** The fact that a certain string of symbols has appeared can aid in predicting which symbol will come next. For instance, if the letters `compr` appear in this article, there is a strong probability they will be followed by an `e`. The PPM scheme maintains such context information to estimate the probability of the next input symbol to appear. An arithmetic coder can use this stream of probabilities to code the source efficiently. Clearly, longer contexts will improve the probability estimation, but require more time to arise. (this is similar to the startup effect in LZ78). To account for this, “escape symbols” exist to progressively step down to shorter context lengths. This introduces a tradeoff in which encoding a long series of escape symbols can require more space than is saved by the use of large contexts. Much effort has gone into choosing probabilities for the escape symbols to minimize their overhead. Storing and searching through each context accounts for the large memory requirements of PPM schemes. PPMD is a recent implementation of the PPM algorithm. Windows users may unknowingly be using PPMD as it is the text compression engine in the popular WinRAR program.

- **Burrows-Wheeler Transform—BWT.** The newest technique among those examined, the Burrows-Wheeler Transform, converts a block of length *n* into a pair consisting of a permutation of *S* (call it *L*) and an integer in the interval `[0...n – 1]`. Though the transformation is simple to describe, it is not obvious how it may be reversed. In latency-critical single-threaded applications, the block-based processing of BWT could be a bottleneck. Several distinct operations must be performed in series (transform, move to front, run-length encode, entropy coding) and entire blocks of data must be processed before moving on to the next. Sorting is the critical operation. Although BWT-based compression could be performed in very little memory with in-place sorting, common implementations use fast sort algorithms and/or structures such as the suffix tree which require substantial memory to provide speed.

The original Lempel-Ziv-inspired methods have remained popular since their newer competitors require more time and memory to achieve compression. PPM variants have been recognized as the leader in compression ratios since their introduction in 1984, but these ratios come at a tremendous time and memory expense. BWT has grown in popularity because its implementations, based on efficient sorting, lead to greater speed than PPM implementations while giving similar excellent compression ratios. Recently, BWT has been recast as a problem similar to PPM, inspiring PPM programs to exploit advances in BWT implementations. It has taken nearly 20 years for implementations of PPM to approach the speed of the LZ77, LZ78, and BWT methods. A compression algorithm may be implemented with many different, yet reasonable, data structures (including binary tree, splay tree, trie, hash table, and list) and yield vastly different performance results.
The quality and applicability of the implementation is as important as the underlying algorithm. We will propose lossless compression techniques for energy savings in ad-hoc network.

Future Work

The lossless compression of the data for wireless transmission for energy saving may be further achieved by developing new framework for evaluating the performance of various compression techniques. By proposing new lossless data compression technique for energy saving, the objective of energy efficient wireless transmission could be achieved. The paper suggests develop a new algorithms for future research in the area of data compression, which can provide better end results as achieve with the implementation of BWT method. The future work will help the wireless services providers and technicians to improvise the latest innovations in the fields of wireless communication based on the latest high speed networks. Future work in this area should examine sensitivity to the type of data. If one knows a priori that data is uncompressable (or can determine this fact dynamically), it is likely to change one’s choice of compression schemes. Sensitivity to the latency requirements of a given task are crucial as well. The results presented in this work are most applicable to the transfer of large files for which one may be willing to tolerate latency. Interactive work requires elimination of perceived delay, and short real-time messages are unlikely to compress well unless they are correlated to provide extensive history. Thus, algorithms which require long warm-up times or large history structures are not likely to be useful.

Conclusion

Optimizing an entire network of devices is a possible desire. Perhaps the sender is not a wall-powered server but another handheld device. Perhaps a poor or crowded communication channel limits the size or speed of a transmission. Many combinations exist for which optimal energy and performance points must be found. How collections of devices might find their desired operating point is another area for research. Most importantly, this work reminds hardware and software developers that committing to one particular compression/decompression scheme is unlikely to be wise in terms of energy. As portable, networked, battery powered computers evolve and become more popular, extended battery lives will grow in importance. Evaluation of a platform’s relative component energy can help one choose the most energy aware lossless compression scheme.

References

Barr C. Kenneth “Energy Aware Lossless Data Compression” MS Computer Science and Engineering Thesis, University of Michigan, September 2002